UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jan Škof

Zasnova in izdelava strežnika za učinkovito obveščanje mobilnih aplikacij

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2014

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jan Škof

Zasnova in izdelava strežnika za učinkovito obveščanje mobilnih aplikacij

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Boštjan Slivnik

Ljubljana, 2014

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Zasnova in izdelava strežnika za učinkovito obveščanje mobilnih aplikacij

Tematika naloge:

Pri uporabi potisnih obvestil mobilni aplikaciji ni potrebno ves čas preverjati, ali ji je bilo neko obvestilo poslano, saj operacijski sistem aplikacijo o prejetju potisnega obvestila avtomatsko obvesti in ji omogoči takojšnje reagiranje na prejeto potisno obvestilo. S tem se na mobilni napravi prihrani računska moč in s tem energija. Izdelajte načrt za strežnik, ki omogoča pošiljanje potisnih obvestil, in ga realizirajte.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Jan Škof, z vpisno številko **63060334**, sem avtor diplomskega dela z naslovom:

Zasnova in izdelava strežnika za učinkovito obveščanje mobilnih aplikacij.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Boštjana Slivnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu prek univerzitetnega spletnega arhiva.

V Ljubljani, dne 19. septembra 2014 Podpis avtorja:

Za pomoč in nasvete pri izdelavi diplomskega dela se zahvaljujem mentorju doc. dr. Boštjanu Slivniku.

Kazalo

Seznam uporabljenih kratic

Kratica	Angleško	Slovensko
GCM	Google Cloud Messaging	Googlova storitev za pošiljanje potisnih obvestil
WNS	Windows Notification Services	Microsoftova storitev za pošiljanje potisnih obvestil
Windows SDK	Windows Software Development Kit	razvojni komplet za Windows in Windows Phone
JSON	JavaScript Object Notation	objektna notacija JavaScript
MVC	Model View Controller	model pogled krmilnika
XML	Extensible Markup Language	razširljivi označevalni jezik
Android SDK	Android Software Development Kit	razvojni komplet za Android
REST	Representational State Transfer	skupina arhitekturnih omejitev

Povzetek

V diplomskem delu sta predstavljena zasnova in razvoj strežnika za učinkovito obveščanje mobilnih aplikacij. Strežnik za učinkovito obveščanje mobilnih aplikacij je sestavljen iz administracijskega dela in dela, namenjenega obveščanju mobilnih aplikacij. Trenutna rešitev omogoča obveščanje mobilnih aplikacij Android in Windows Phone, ne omogoča pa obveščanja mobilnih aplikacij, ki temeljijo na priljubljenem operacijskem sistemu Ios. Strežnik je zasnovan tako, da ga je mogoče nadgraditi tudi z ostalimi operacijskimi sistemi, kot so Ios, Firefox OS itd. Poleg možnosti nadgradnje na ostale operacijske sisteme omogoča nadgradnjo na klasične sisteme obveščanja, kot so SMS in elektronska pošta.

Ključne besede: GCM, obvestilo, WNS, obveščanje, Android, Windows Phone

Abstract

This thesis presents the design and implementation of a server for efficient notification of mobile applications. The server consist of two parts. The first part is dedicated for sending push notifications to mobile applications, the second part is dedicated for the administration of mobile applications. The current implementation allows sending push notifications to Android and Windows Phone enabled devices. The server is designed in a way to allow future upgrades for sending Ios or Firefox push notifications. Inaddition to upgrades on other operating system it allows ugrades to classic notification system like email or SMS.

Keywords: GCM, notification, WNS, informing, Android, Windows Phone

Poglavje 1 Uvod

V zadnjih letih postajajo pametni telefoni čedalje bolj razširjeni med prebivalstvom. Njihova množična uporaba predstavlja novo dimenzijo komunikacije in interakcije med človekom in tehnologijo.

Pred leti so telefoni omogočali obveščanje in komunikacijo le prek kratkih tekstovnih obvestil, imenovanih SMS. Z uporabo interneta sta se obveščanje in komunikacija razširila na elektronsko pošto, kjer je bilo možno prejemati ne le kratka tekstovna obvestila, temveč tudi slike in razne dokumente. Z razvojem in uporabo pametnih telefonov se je uveljavila nova tehnologija obveščanja, imenovana potisna obvestila. Ta obvestila omogočajo poleg kratkih tekstovnih obvestil in slik tudi razne akcije, ki so povezane s potisnim obvestilom, kot so na primer prikaz videovsebin, spletnih mest, zagon mobilne aplikacije in v njej prikaz določenih vsebin ter veliko drugih stvari.

Potisna obvestila so bila najprej razvita pri podjetju Apple Inc. kot alternativa tehnologijam oziroma aplikacijam, ki so periodično preverjala nova obvestila. Periodično preverjanje novih obvestil je pomenil veliko porabo baterije samega telefona in veliko število prenešenih ter poslanih podatkov. Tehnologija potisnih obvestil deluje na obratnem načelu. Aplikacija se mora najprej registrirati pri ponudniku potisnih obvestil in pridobiti enoličen identifikator, ta pa služi za prepoznavanje pametnega telefona in mobilne aplikacije. Vsak telefon ima lahko več enoličnih identifikatorjev pri enakem ponudniku potisnih obvestil (Google Cloud Messaging, Windows Notification Service itd.) za vsako nameščeno aplikacijo na pametnem telefonu. Po uspešni registraciji se vzpostavi povezava med ponudnikom potisnih obvestil in pametnim telefonom, kjer telefonu oziroma mobilni aplikaciji ni treba periodično preverjati novih obvestil. Ob prispetju novega obvestila ponudnika potisnih obvestil z uporabo enoličnega identifikatorja posreduje potisno obvestilo pametnemu telefonu oziroma mobilni aplikaciji, nameščeni na pametnem telefonu.

V diplomskem delu sta predstavljena zasnova in razvoj strežnika za učinkovito obveščanje mobilnih aplikacij. Strežnik za učinkovito obveščanje mobilnih aplikacij deluje v okviru dveh operacijskih sistemov Android in Windows Phone. Trenutna rešitev ne omogoča pošiljanja priljubljenemu operacijskemu sistemu Ios. Strežnik je zasnovan tako, da ga je mogoče nadgraditi tudi z ostalimi operacijskimi sistemi, kot so Ios, Firefox OS itd. Poleg možnosti

nadgradnje na ostale operacijske sisteme omogoča nadgradnjo na klasične sisteme obveščanj, kot sta SMS in elektronska pošta.

V drugem poglavju bomo predstavili razvoj rešitve. Podrobno bomo predstavili tako uporabljene tehnologije za razvoj strežniškega dela kot tudi razvoj odjemalčevega dela. Strežniški del je odgovoren za pripravo in učinkovito pošiljanje potisnih obvestil do ponudnikov potisnih obvestil. Pri strežniškem delu so ključnega pomena ponudniki potisnih obvestil. Predstavljena bosta Google Cloud Messaging in Windows Notifiction Services kot dva ponudnika potisnih obvestil ter njune ključne lastnosti.

V tretjem poglavju se bomo dotaknili testiranja programske rešitve. Dobri rezultati testiranja programske rešitve predstavljajo dobre možnosti za uspeh programske rešitve na svetovnem trgu mobilnih aplikacij. Predstavili bomo rezultate meritev, ki smo jih dobili pri pošiljanju 10.000, 100.000 in 1.000.000 potisnih obvestil. Predstavljate si pošiljanje potisnih obvestil mobilni aplikaciji, kot je Talking Tom Cat, ki ima milijone uporabnikov na več različnih operacijskih sistemih.

Diplomsko delo bomo zaključili s predstavitvijo možnih načinov uporabe potisnih obvestil v realnem svetu.

Poglavje 2 Razvoj programske rešitve

Razlog za razvoj strežnika za učinkovito obveščanje mobilnih aplikacij je bilo pomanjkanje funkcionalnosti pri obstoječih rešitvah. Obstoječe rešitve, kot so Urban Airship [1], Push.IO [2], ZeroPush [3] itd., omogočajo le osnovne funkcije potisnih obvestil, kot so osnovna postavitev in oblika potisnega obvestila, potisnemu obvestilu ni možno dodati akcije, s katero bi lahko sprožili dodatne video- ali druge vsebine. V veliki večini primerov podpirajo le en operacijski sistem. Z razvojem strežnika za učinkovito obveščanje mobilnih aplikacij bi se radi izognili naštetim pomanjkljivostim trenutnih rešitev in tako na enem mestu podprli več različnih operacijskih sistemov ter vse njihove funkcionalnosti.

2.1 Načrt programske rešitve

Za razvoj in testiranje programske rešitve je bilo treba razviti tri aplikacije, ki se med seboj uspešno povezujejo. Razviti je bilo treba mobilno aplikacijo, strežnik za učinkovito obveščanje mobilnih aplikacij in spletno aplikacijo za administracijo mobilnih naprav ter obvestil. Mobilna aplikacija teče na pametnemu telefonu. Razviti sta bili mobilni aplikaciji za operacijska sistema Android in Windows Phone. Mobilni aplikaciji se morata povezati do ponudnika potisnih obvestil, pridobiti enolični identifikator in ga posredovati do strežnika za učinkovito obveščanje mobilnih aplikacij. Enolični identifikator služi za prepoznavanje mobilnih aplikacij na pametnih telefonih. Ta telefon prek enoličnega identifikatorja posreduje potisno obvestilo mobilni aplikaciji, ki ji je namenjeno potisno obvestilo. Na tak način se izognemo možnosti, da bi potisno obvestilo prejela druga mobilna aplikacija oziroma pametni telefon. Potisna obvestila so posredovana do mobilne aplikacije prek ponudnikov potisnih obvestil. Strežnik za učinkovito obveščanje mobilnih aplikacij je implementiran kot servis, ki skrbi za učinkovito obveščanje mobilnih aplikacij. Za posredovanje obvestil do ponudnika potisnih obvestil se mora strežnik registrirati pri ponudniku in pridobiti varnostni žeton, ki ga vključi v obvestilo pri vsakem poslanem obvestilu. Strežnik posreduje obvestila v najkrajšem možnem času ponudniku potisnih obvestil, ki potisne obvestilo do mobilne aplikacije prek varne povezave. Komunikacija med strežnikom in ponudnikom potisnih obvestil poteka prek varne povezave. Poleg servisa je strežnik sestavljen iz spletne aplikacije, ta pa skrbi za registracijo mobilnih naprav, generiranje in pregledovanje obvestil.

Za učinkovito obveščanje mobilnih aplikacij je bilo treba strežnik preveriti, kako se odreže pri obveščanju prek 1.000.000 mobilnih aplikacij. Strežniku je bilo posredovano veliko število obvestil, ta pa je bilo treba posredovati v najkrajšem možnem času. Merjenje časa pošiljanja je začelo teči, ko je bilo dodano prvo obvestilo v vrsto za pošiljanje, dokler vrsta za pošiljanje ni bila prazna.

2.1.1 Tehnologije

Servis za pošiljanje potisnih obvestil je razvit v okolju .NET različice 4.5.1. v programskem jeziku C# [4]. Razvit je v obliki Windows servisa, ki ga avtomatsko zažene in opravlja operacijski sistem. Za komunikacijo z ostalimi sistemi in procesi uporablja tehnologijo Windows Comunication Fundation – WCF [5], ki omogoča enostavno komunikacijo prek protokola SOAP. Za hkratno pošiljanje potisnih obvestil je uporabljena knjižnica Task Parallel Library – TPL [6], ki omogoča delo z nitmi.

Strežniška aplikacija za upravljanje potisnih obvestil je razvita v okolju .NET različice 4.5.1 z uporabo ogrodja ASP.NET MVC 4 in jezika C# [7]. ASP.NET MVC je ogrodje, ki uporablja pristop Model-View-Controller – MVC – za gradnjo same arhitekture aplikacije. Taka arhitektura aplikacije omogoča ločitev posameznih delov aplikacije, kot sta uporabniški vmesnik in poslovna logika. Strežniška aplikacija je zgrajena iz dveh delov; dela, ki je namenjen uporabnikom, in dela, ki je namenjen komunikaciji z mobilnimi napravami. Uporabniški vmesnik, namenjen uporabniku, je razvit v označevalnem (markup) jeziku Razor. Deli, namenjeni izključno komuniciranju med pametnim telefonom in aplikacijo za upravljanje, so razviti v obliki servisov REST s pomočjo tehnologije ASP.NET WEB API. Prenos podatkov med servisi je v obliki notacije JSON.

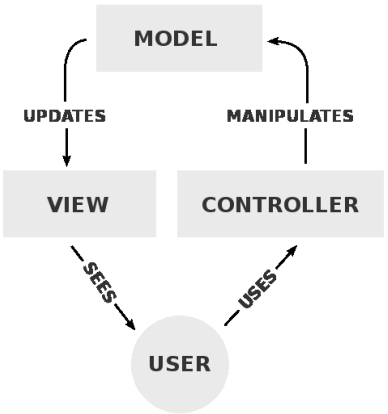
Odjemalska aplikacija, razvita na Microsoftovi platformi Windows Phone, je razvita za verzijo 8.1. Aplikacija je razvita v tehnologiji Silverlight [8] v programskem jeziku C#. Komunikacija z zalednim strežnikom poteka prek protokola http, klici so asinhroni, uporabljena je tehnologija asynac/await [9], saj na takšen način omogočimo najboljšo uporabniško izkušnjo, ker ne prihaja do zamrznitve uporabniškega zaslona.

Odjemalska aplikacija, razvita na Googlovi platformi Android [10], je razvita za verzijo 4.4.0 KitKat. Aplikacija je razvita v tehnologiji Java s pomočjo Android Studia. Enako kot pri prvi aplikaciji za Windows Phone poteka vsa komunikacija asinhrono za najboljšo uporabniško izkušnjo.

2.1.2 Vzorci programiranja

Strežnik za učinkovito obveščanje mobilnih aplikacij je implementiran na podlagi vzorca proizvajalec in potrošnik [11] (angl. *producer consumer pattern*) za del, ki je namenjen obveščanju mobilnih aplikacij – servis. Problem proizvajalca in potrošnika je v računalništvu klasičen problem sinhronizacije med procesi ali nitmi. Problem opisuje dva procesa ali niti, proizvajalca in potrošnika, ki si izmenjujeta skupni pomnilnik (angl. *shared memory*) v obliki vrste (angl. *queue*). Naloga proizvajalca je generiranje podatkov, ki jih vstavlja v vrsto, hkrati pa je naloga potrošnika obdelati te podatke in jih jemati ven iz vrste. Paziti je treba, da se vrste ne napolni preveč s hkratnimi podatki in da potrošnik ne jemlje podatke iz prazne vrste. Rešitev za proizvajalca v primeru prenapolnjene vrste je faza mirovanja, dokler potrošnik ne izprazni vrste ali vsaj ne zmanjša količine podatkov v vrsti. V primeru zmanjšanja podatkov v vrsti mora potrošnik signalizirati proizvajalcu, da lahko prekine fazo mirovanja. Signalizacija lahko poteka prek semaforjev, monitorjev itd. Površna implementacija proizvajalca in potrošnika lahko pripelje do zastoja med procesi ali nitmi, kjer oba procesa čakata drug na drugega.

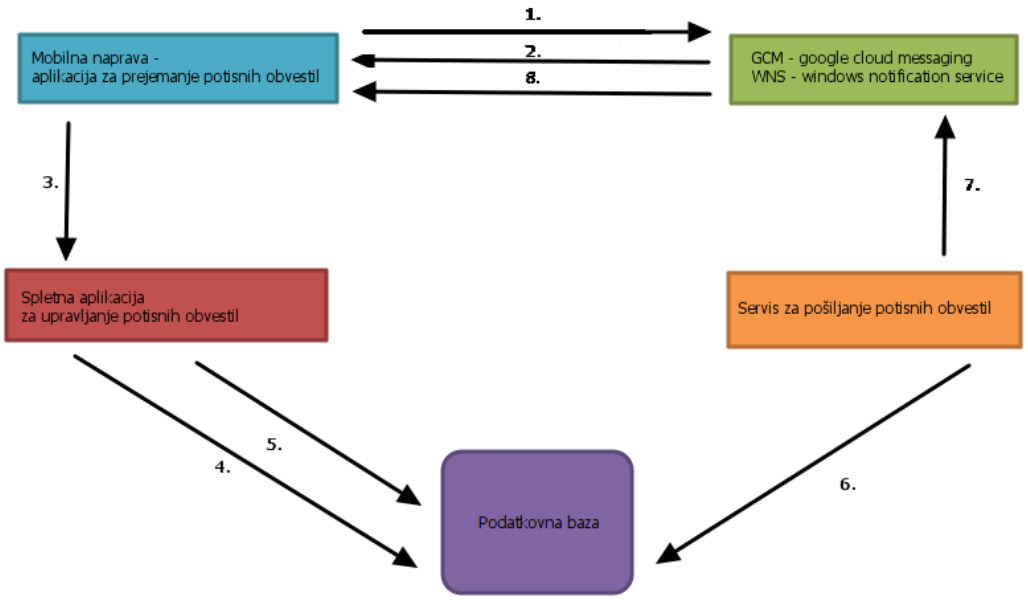
Spletna aplikacija, katere namen je registracija mobilnih naprav, generiranje in pregledovanje obvestil, je implementirana na podlagi pristopa MVC (Model View Controller). Pristop MVC je namenjen gradnji uporabniških vmesnikov. Pristop razdeli aplikacijo na tri med seboj povezane dele, tako da se loči interna predstavitev informacij od samega uporabniškega vmesnika. Primarni del pristpa MVC je model, ki zajema predstavitev problemske domene v aplikaciji ne glede na uporabniški vmesnik. View je lahko katera koli izhodna oblika informacije, ki jo predstavlja model, in je lahko v obliki grafa ali besedila. Tretji del, imenovan controller, sprejema vhodne podatke, ki jih posreduje modelu prek ukazov.



Slika 2.1: Prikaz vzorca MVC

2.1.3 Proces komunikacije

Splošen proces komunikacije med aplikacijami je ponazorjen na spodnji sliki:



Slika 2.2: Proces komunikacije

1. Aplikacija pošlje zahtevo za registracijo potisnih obvestil,
2. ponudnik potisnih obvestil registrira aplikacijo tako, da generira enoličen identifikator, ki ga posreduje aplikaciji,
3. aplikacija posreduje enolični identifikator strežniku za učinkovito obveščanje mobilnih aplikacij,
4. identifikator se shrani v bazo podatkov,
5. generiranje potisnega obvestila in določanje prejemnikov potisnega obvestila,
6. branje potisnega obvestila,
7. pošiljanje potisnega obvestila ponudniku potisnih obvestil,
8. prejemanje in prikazovanje potisnega obvestila na mobilnem telefonu.

2.2 Odjemalec

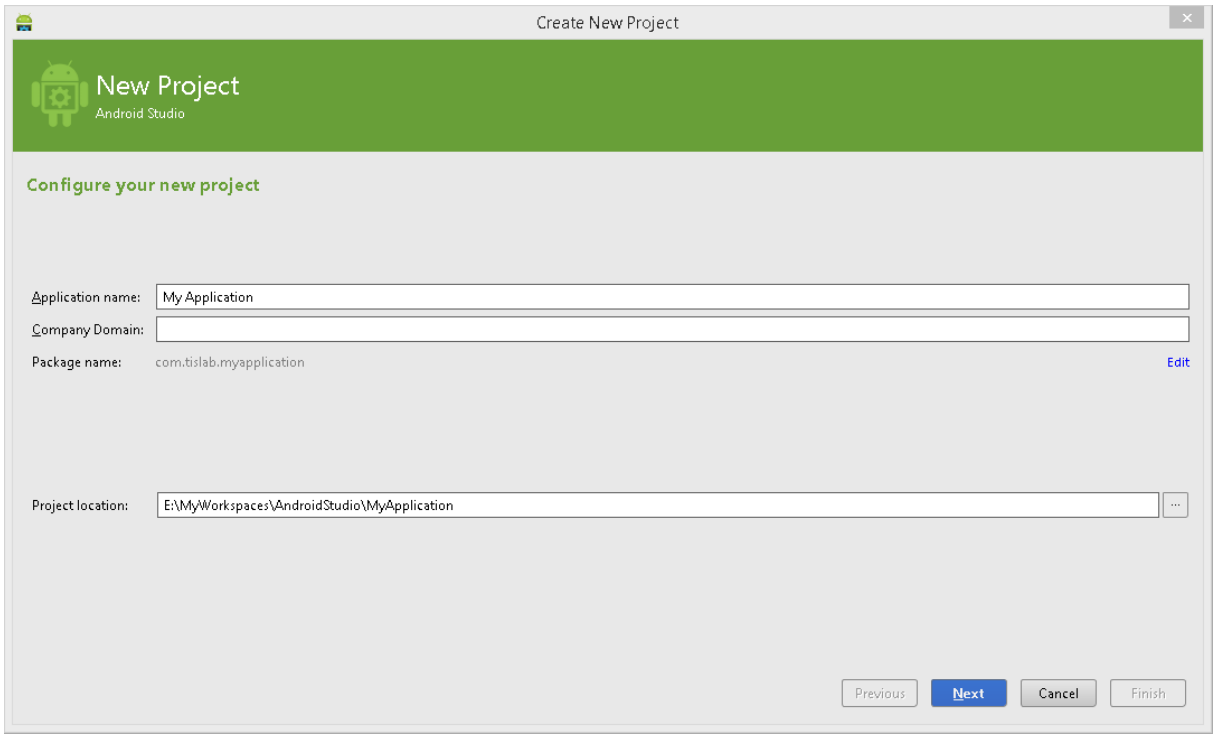
Model odjemalec strežnik je oblika porazdeljene zasnove računalniške arhitekture. Taka arhitektura loči ponudnike virov, imenovane strežniki, od potrošnikov, imenovane odjemalci. Komunikacija pogosto poteka prek računalniškega omrežja, ki je najpogosteje internet, lahko pa je tudi mobilno telefonsko omrežje. Na strežniku lahko teče več aplikacij, ki delijo njihove vire z odjemalci, ti pa ne delijo nobenih virov, temveč le zahtevajo vire od strežnikov.

2.2.1 Android

Preden lahko začnemo z delom na platformi Android, si je treba zagotovit delovno okolje. Treba je imeti nameščeno:

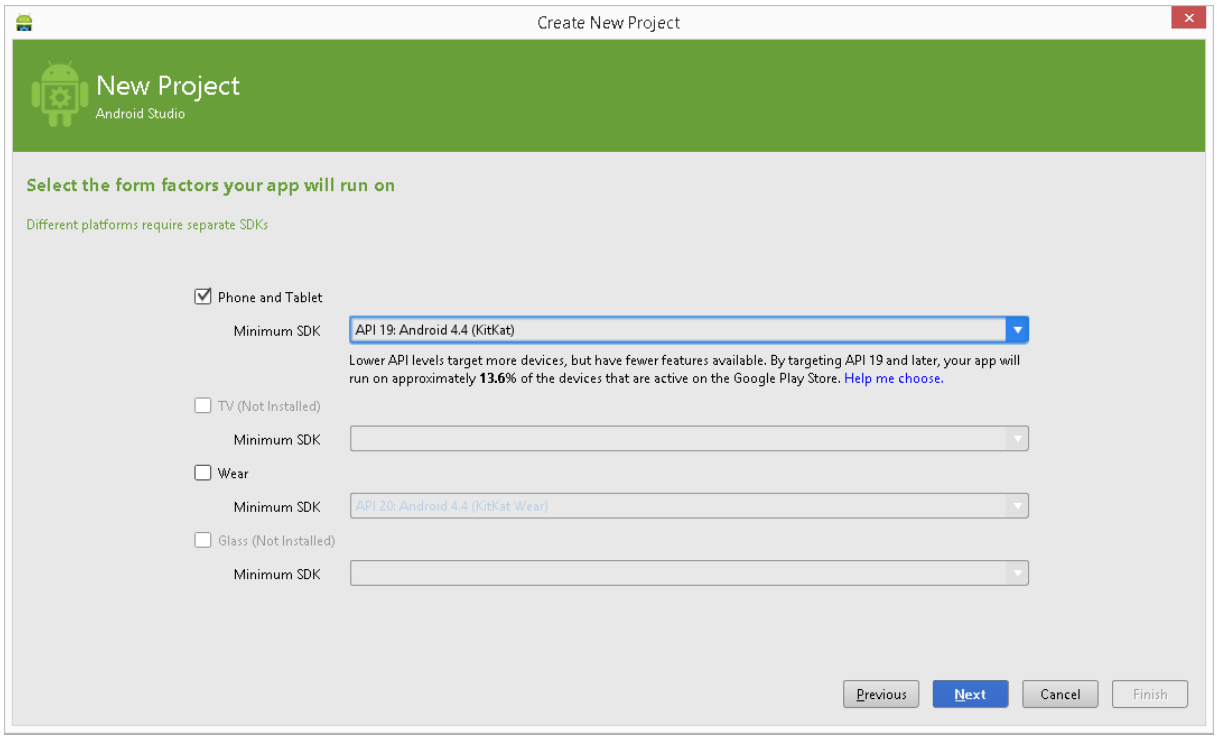
- Android kit za razvoj programske opreme (Android SDK [12]),
- integrirano razvojno orodje, kot sta na primer Android Studio [13] ali Eclipse ADT,
- in ostala orodja, potrebna za razvoj na platformi Android (vtič za razhroščevanje itd.).

Za začetek razvoja v Android Studio moramo najprej kreirati projekt, ki bo vseboval vse naše izvorne datoteke.



Slika 2.3 Kreiranje novega projekta Android

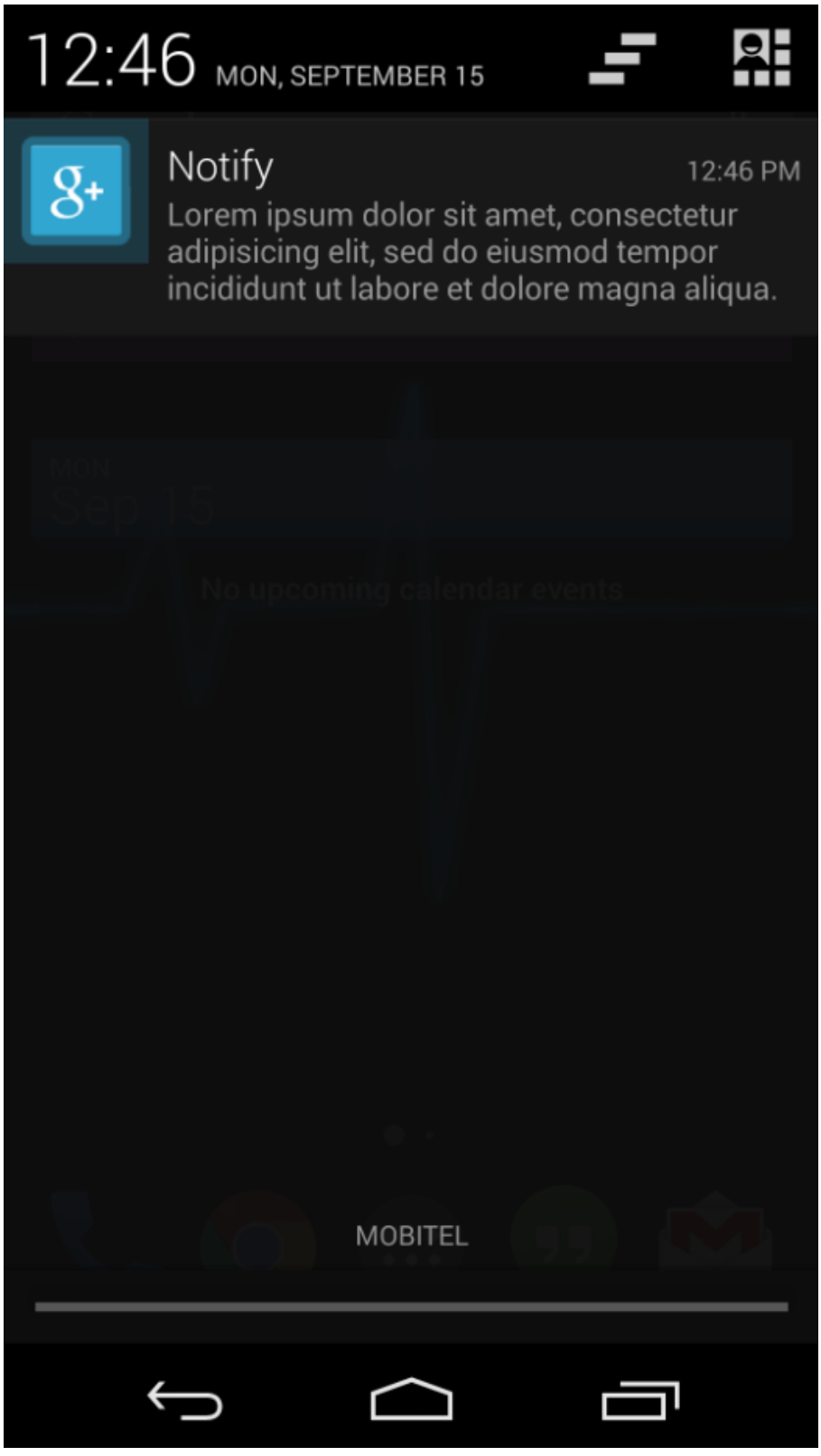
Določiti je treba verzijo operacijskega sistema Androida.



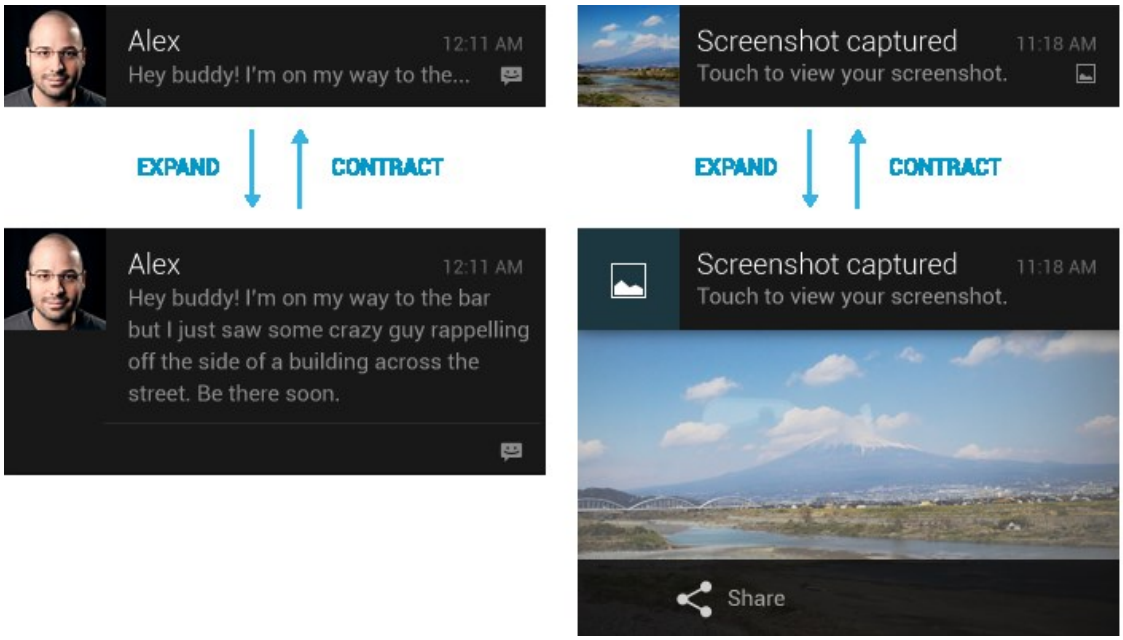
Slika 2.4: Določanje verzije operacijskega sistema Android

Android je namenil večji pomen na potisna obvestila z verzijo 4.0 Jelly Bean [14]. Omogočil je razširjenje potisnih obvestil za prikazovanje dodatnih informacij, na potisna obvestila je možno obvestiti akcije, ki ob kliku uporabnika sprožijo določen dogodek. Potisna obvestila lahko tudi razvrščamo po prioriteti, in ne samo po času nastanka.

Zgradba potisnih obvestil je lahko osnovne oblike ali razširjene oblike. Osnovna oblika vsebuje sliko ali ikono pošiljatelja, naslov, besedilo in datum kreiranja potisnega obvestila. Razširjena oblika omogoča daljša besedila, na tak način lahko prejemnik prebere celotno obvestilo. Razširjena potisna obvestila omogočajo prikazovanje večjih slik.

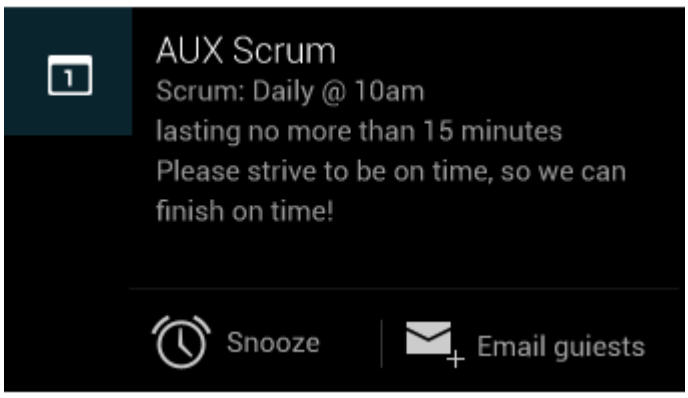


Slika 2.5: Prikaz osnovne oblike potisnega obvestila



Slika 2.6: Prikaz razširjene oblike potisnega obvestila

Akcije, ki so bile dodane z verzijo 4.0 Jelly Bean in prikazane na dnu potisnega obvestila, omogočajo hitre akcije, kot je pošiljanje elektronske pošte, ne da bi bilo treba odpreti aplikacijo za pošto. Pri dodajanju akcij je treba biti pazljiv, saj ne želimo povečati zapletenosti potisnih obvestil.



Slika 2.7: Prikaz akcij na potisnem obvestilu

Potisna obvestila ni primerno pošiljati v primerih, ko:

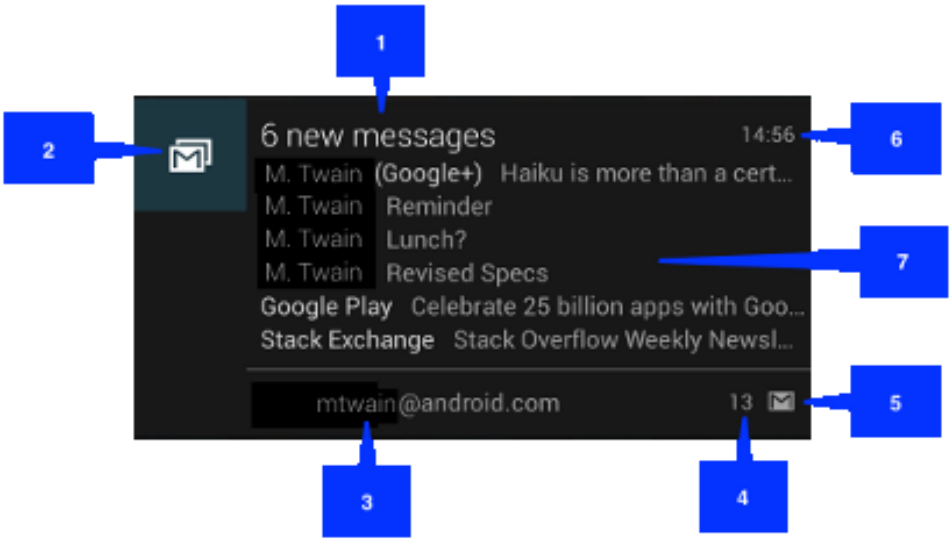
- vsebina ni dosledno vezana na uporabnika,
- uporabnik pregleduje obvestilo v aplikaciji,
- prihaja do napak, ki jih lahko aplikacija sama odpravi,
- želimo, da uporabnik z njo zažene aplikacijo,
- želimo promovirati svojo lastno blagovno znamko.

Elementi potisnih obvestil so naslov (1), slika (2), besedilo (3), dodatne informacije (4), ikona aplikacije (5), čas prejetja obvestila (6) in pri razširjeni obliki možnost daljšega besedila ali večje slike (7).

Osnovna oblika:

Slika 2.8: Prikaz elementov osnovne oblike potisnega obvestila

Razširjena oblika:



Slika 2.9: Prikaz elementov razširjene oblike potisnega obvestila

Razširjena oblika je popolnoma enaka osnovni obliki, razen pri točki sedem, kjer je možno prikazati dodatno besedilo in ga tudi oblikovati.

Kreiranje potisnega obvestila

Definicijo uporabniškega vmesnika (naslova, besedila, slike) in akcij se definira prek razreda NotificationCompact.Builder. Za kreiranje osnovnega potisnega obvestila je dovolj poklicati metodo build() znotraj razreda NotificationCompact.Builder, ki vrne objekt razreda Notification. Za prikaz potisnega obvestila je treba poklicati sistemsko metodo NotificationManager.notify().

Naslov, besedilo in sliko se lahko definira prek metod setSmallIcon(), setContentTitle() in setContectText().

Akcije, ki jih je možno pripeti potisnemu obvestilu, niso obvezne, obvezna je le osnovna akcija, ki uporabnika preusmeri na predefiniran uporabniški vmesnik znotraj aplikacije. Potisno obvestilo lahko zagotovi več akcij z dodajanjem gumbov na samo potisno obvestilo. Možnost dodajanja gumbov na potisno obvestilo je možno le od verzije 4.1. Akcija znotraj potisnega obvestila je definirana prek razreda PendingIntent. Na primer, če želimo odpreti določen uporabniški vmesnik znotraj aplikacije, ob kliku na besedilo potisnega obvestila pokličemo metodo setContentIntent().

```
private void sendNotification(String msg) {

    mNotificationManager = (NotificationManager)
            this.getSystemService(Context.NOTIFICATION_SERVICE);

    PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
            new Intent(this, MainActivity.class), 0);

    NotificationCompat.Builder mBuilder =
            new NotificationCompat.Builder(this)
                    .setSmallIcon(R.drawable.common_signin_btn_icon_pressed_light)
                    .setContentTitle("Title")
                    .setStyle(new NotificationCompat.BigTextStyle())
                    .setContentText("Content");

    mBuilder.setContentIntent(contentIntent);

    mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
}
```

Slika 2.10: Implementacija potisnega obvestila

Pri posodabljanju potisnih obvestil se je dobro izogibati ponovnemu kreiranju potisnega obvestila. Posodabljanje potisnih obvestil, ki so že bila posredovana sistemu, je najbolje posodabljati prek NOTIFICATION_ID, ki je bil posredovan sistemski metodi za prikazovanje potisnega obvestila.

Potisna obvestila ostajajo vidna, dokler se ne zgodi eden od naslednjih dogodkov:

- uporabnik izbere potisno obvestilo. V tem primeru sistem pokliče metodo setAutoCancel(),
- aplikacija pokliče metodo cancel() za določen NOTIFICATION_ID,
- aplikacija pokliče metodo cancelAll().

GCM – Google Cloud Messaging

GCM [15] je servis, ki omogoča pošiljanje in prejemanje podatkov od uporabnikov operacijskega sistema Android, skrbi za vso sinhronizacijo in dostavo potisnih obvestil do samega prejemnika. Servis GCM nima nobenih omejitev glede števila potisnih obvestil in je popolnoma brezplačen.

Lastnosti servisa GCM:

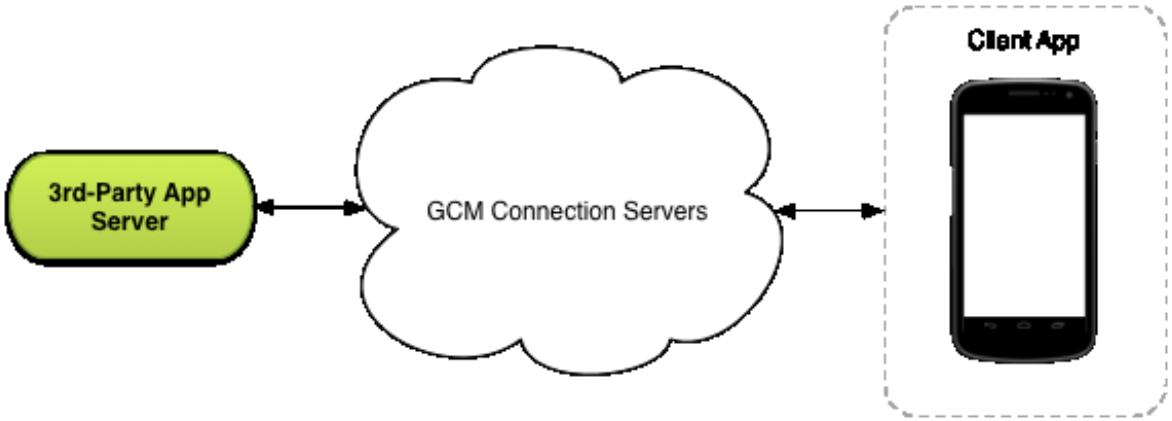
- omogoča pošiljanje potisnih obvestil iz aplikacijskih strežnikov do samih odjemalcev,
- z uporabo GCM Connector Server lahko poteka komunikacija tudi dvosmerno,
- aplikacijo ni treba zagnati znotraj sistema Android, da bi lahko prejela potisno obvestilo.
- GCM pošlje le podatke do sistema Android, aplikacija je zadolžena za prikaz le-teh,
- za nemoteno delovanje GCM je zahtevan Android 2.2, ki ima nameščeno aplikacijo Google Play Store,
- uporablja obstoječo povezavo za prejemanje potisnih obvestil od Google Services. Za operacijske sisteme, novejše od različice 4.0.4, ni treba nastavljati Googlovega računa.

Za uspešno izvedbo vseh korakov pri pošiljanju potisnih obvestil sta potrebna dva ključna pogoja. Izpolnjevati je treba varnostne pogoje, ki so vključeni na različnih fazah komunikacije z Google Cloud Messaging. Tako želi Google zagotovili varnost vseh vključenih strank. V proces komunikacije je treba vključiti vse subjekte, ki igrajo ključno vlogo pri pošiljanju in prejemanju potisnih obvestil.

Pri varnostnih pogojih je treba zagotoviti:

- ID pošiljatelja – identifikator, s katerim se avtentificiramo pri servisih GCM,
- ID aplikacije – identifikator aplikacije za pravilno dostavljanje potisnega obvestila,
- ID registracije – identifikator za pošiljanje potisnih obvestil, izdan s strani servisa GCM,
- žeton pošiljatelja – žeton, ki omogoča aplikacijskemu strežniku dostop do servisov GCM.

Pri procesu komunikacije je treba vključiti mobilno aplikacijo, strežniško aplikacijo in Google Cloud Messaging. Mobilna aplikacija mora biti nameščena na telefonu z operacijskim sistemom Android, ki podpira vsaj verzijo 2.2, nameščeno mora imeti pa tudi aplikacijo Google Play Store. Strežniška aplikacija mora omogočati pošiljanje podatkov do telefona prek servisa Google Cloud Messaging. Strežniška aplikacija mora znati prejemati podatke od mobilne aplikacije. GCM Connection Servers imajo vlogo dostave potisnih obvestil od strežniške aplikacije do mobilne aplikacije na pametnem telefonu.



Slika 2.11: Arhitektura storitve GCM

Google ponuja GCM Connection Servers, ki prejemajo potisna obvestila. Prejeta obvestila posredujejo mobilnim aplikacijam, ki so registrirane kot veljavne aplikacije GCM,

- aplikacijski strežnik je komponenta oz. servis, ki komunicira s servisi GCM. Servis generira potisna obvestila, ki jih posreduje servisu GCM,
- mobilna aplikacija, ki je registrirana pri servisih GCM.

Mobilna aplikacija se najprej registrira pri servisih GCM, tako da prejme enoličen identifikator. Registracija je prikazana na sliki 2.12.

```
private void registerInBackground() {
    new AsyncTask<Void, Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
            String msg = "";
            try {
                if (gcm == null) {
                    gcm = GoogleCloudMessaging.getInstance(context);
                }
                regid = gcm.register(SENDER_ID);
                msg = "Device registered, registration ID=" + regid;

                sendRegistrationIdToBackend(regid);

                storeRegistrationId(context, regid);
            } catch (IOException ex) {
                msg = "Error :" + ex.getMessage();
            }
            return msg;
        }

        @Override
        protected void onPostExecute(String msg) {

            TextView tv = (TextView) findViewById(R.id.tvNotificationId);
            tv.setText(msg);
        }
    }.execute(null, null, null);
}
```

Slika 2.12: Implementacija registracije mobilne aplikacije

Pridobljen identifikator je treba po varni poti posredovati aplikacijskemu strežniku. V primeru prestrezanja identifikatorja s strani nepooblaščene osebe lahko pripelje do zlorabe. Uporabnike mobilne aplikacije lahko obveščajo nepooblaščeni servisi oziroma osebe. V spodnjem primeru je razvidno, da za prenos identifikatorja do aplikacijskega strežnika ni uporabljena nobena storitev za njegov varen prenos. V produkcijskih okoljih je treba vsaj identifikator kriptirati z eno od kriptografskih funkcij, s katero bi preprečili nepooblaščenim osebam uporabo identifikatorja. Na drugi strani bi moral aplikacijski strežnik poznati način, kako dekriptirati dobljeni identifikator.

```
private void sendRegistrationIdToBackend(String regId) {
    new AsyncTask<String, Void, String>() {

        @Override
        protected String doInBackground(String... params) {

            String postText = "No response from server";
            try {
                //StringEntity sendStuff = new StringEntity(params[0]);

                HttpClient httpClient = new DefaultHttpClient();
                HttpPost post = new HttpPost("http://93.103.9.79:3333/api/device?regId="
                        +params[0]+"&os=Android&osVersion=4.4.2");
                post.setHeader("Accept", "application/json");
                post.setHeader("Content-type", "application/json");
                post.setHeader("Accept-Encoding", "gzip");
                //post.setEntity(sendStuff);

                HttpResponse postResponse = httpClient.execute(post);
                postText = EntityUtils.toString(postResponse.getEntity());

            } catch (IOException e) {
                e.printStackTrace();
            }
            return postText;
        }

        protected void onPostExecute(String msg) {

            TextView tv = (TextView) findViewById(R.id.tvServerResponse);
            tv.setText(Html.fromHtml(msg));
        }
    }.execute(regId, null, null);
}
```

Slika 2.12: Implementacija pošiljanja registracijskega identifikatorja

Aplikacijski strežnik pošlje potisno obvestilo do servisov GCM, ki posredujejo potisno obvestilo do mobilne aplikacije. Operacijski sistem Android prejme surovo vsebino potisnega obvestila, ki jo posreduje do mobilne aplikacije prek namena com.google.android.c2dm.intent.RECEIVE v naboru dodatnih podatkov. Mobilna aplikacija izvleče iz namena podatke, ki jih oblikuje in prikaže na uporabniškem zaslonu.

```
protected void onHandleIntent(Intent intent) {
    Bundle extras = intent.getExtras();
    GoogleCloudMessaging gcm = GoogleCloudMessaging.getInstance(this);
    String messageType = gcm.getMessageType(intent);

    if (!extras.isEmpty()) {
        if (GoogleCloudMessaging.
                MESSAGE_TYPE_SEND_ERROR.equals(messageType)) {
            sendNotification("Send error: " + extras.toString());
        } else if (GoogleCloudMessaging.
                MESSAGE_TYPE_DELETED.equals(messageType)) {
            sendNotification("Deleted messages on server: " +
                    extras.toString());
        } else if (GoogleCloudMessaging.MESSAGE_TYPE_MESSAGE.equals(messageType)) {
            sendNotification("Received: " + extras.toString());
        }
    }
    GcmBroadcastReceiver.completeWakefulIntent(intent);
}
```

Slika 2.13: Implementacija prejemanja potisnega obvestila

2.2.2 Windows Phone

Za delo na platformi Windows Phone različice 8 potrebujemo:

- operacijski sistem Windows 7, 8.0, 8.1 (x86 ali x64),
- Windows Phone 8.1 kit (gonilniki za emulator),
- Windows Phone SDK 8,
- Windows Driver Kit 8.1 (gonilniki za delo s fizičnim telefonom),
- razvojno ogrodje Visual Studio 2013.

Operacijski sistem Windows Phone omogoča več vrst potisnih obvestil [16]. To so:

- ploščice, ki predstavljajo aplikacijo na začetnem zaslonu,
- značke, ki omogočajo prikaz statusa aplikacije znotraj ploščic, in
- toast obvestila, ki se prikažejo na vrhu zaslona prek upravljavca obvestil.

Za prejemanje potisnih obvestil je treba odjemalčevo aplikacijo nastavit tako, da lahko prejema potisna obvestila. To storimo na naslednji način:

V razredu, ki skrbi za uporabniški vmesnik – v večini primerov je privzeto ime razreda MainPage.xaml.cs –, dodamo referenco na:

```
using Microsoft.Phone.Notification;
using System.Text;
```

in dopolnimo izvorno kodo v konstruktorju, ki bo poskrbela vzpostaviti povezavo do servisa za pošiljanje potisnih sporočil in se uspešno registrirala.

```
public MainPage()
{
 InitializeComponent();

 HttpNotificationChannel pushChannel;
 string channelName = "povezava";

 //preverimo, ali že obstaja obstoječa povezava:
 pushChannel = HttpNotificationChannel.Find(channelName);

 //v primeru null vrednosti obstoječa povezava ne obstaja:
 if (pushChannel == null)
 {
  //kreiramo novo povezavo:
  pushChannel = new HttpNotificationChannel(channelName);

  //pridobimo registracijski ID in ga pošljemo aplikacijskemu strežniku:
 pushChannel.ChannelUriUpdated+=new
 EventHandler<NotificationChannelUriEventArgs>(PushChannel_ChannelUriUpdated);

  //v primeru napak lahko obvestimo uporabnika oz. skrbnika aplikacije:
 pushChannel.ErrorOccurred+=new
 EventHandler<NotificationChannelErrorEventArgs>(PushChannel_ErrorOccurred);

 //izvorna koda, ki se sproži ob prejetju potisnega obvestila:
 //na dogodek se ni treba registrirat, če ne želimo dodatno obdelati potisnega obvestila:
```

```
 pushChannel.ShellToastNotificationReceive+=new
 EventHandler<NotificationEventArgs>(PushChannel_ShellToastNotificationReceived);

  pushChannel.Open();
  pushChannel.BindToShellToast();
 }
 else
 {
  //preverimo, ali se je aplikaciji spremenil registracijski ID:
  //v primeru spremembe tega tudi pošljemo aplikacijskemu strežniku:
  pushChannel.ChannelUriUpdated +=new
  EventHandler<NotificationChannelUriEventArgs>(PushChannel_ChannelUriUpdated);

  // v primeru napak lahko obvestimo uporabnika oz. skrbnika aplikacije:
  pushChannel.ErrorOccurred +=new
  EventHandler<NotificationChannelErrorEventArgs>(PushChannel_ErrorOccurred);

  // izvorna koda, ki se sproži ob prejetju potisnega obvestila:
  //na dogodek se ni treba registrirat, če ne želimo dodatno obdelati potisnega obvestila:
  pushChannel.ShellToastNotificationReceived +=new
  EventHandler<NotificationEventArgs>(PushChannel_ShellToastNotificationReceived);
 }
}
```

Enako, kot velja pri odjemalcu Android, je treba ID registracije poslati do aplikacijskega strežnika. Ta povezava naj bo kriptirana oziroma mora imeli višjo raven varnosti. V primeru prestrezanja ID-ja s strani nepooblaščene osebe lahko pripelje do nezaželenih sporočil in zlorab telefona.

Oblika potisnega obvestila:

```
<?xml version=\"1.0\" encoding=\"utf-8\"?>
<wp:Notification xmlns:wp=\"WPNotification\">
 <wp:Toast>
  <wp:Text1> naslov </wp:Text1>
  <wp:Text2> besedilo </wp:Text2>
  <wp:Param> UI razred (MainPage.xaml) </wp:Param>
 </wp:Toast>
</wp:Notification>
```

WNS – Windows Notification Service

Windows Notification Service [17] in vse pripadajoče knjižnice predstavljajo osnovo za razvoj potisnih obvestil na operacijskem sistemu Windows Phone. Prednosti uporabe WNS-storitev je dostava potisnih obvestil v roku petih sekund, če je naprava priklopljena na internet. Ni potrebnih nobenih certifikatov za avtentifikacijo. WNS je primeren tudi za aplikacije Windows Store.

Za uspešno uporabo WNS-storitev se je najprej treba registrirati kot razvijalec Windows Store. Drugi korak je registracija odjemalske aplikacije na nadzorni plošči Windows Store. Po uspešni registraciji aplikacije se pridobi identifikacijski element, ki ga je treba vključiti v manifest aplikacije. Primer:

```
<Identity Name="NotificationTest.ApplicationTest"
        Version="1.0.0.0"
        Publisher="CN=Microsoft Corporation,
        O=Microsoft Corporation, L=Redmond, S=Washington, C=US" />
```

Tretji korak je registracija aplikacijskega strežnika. Pridobiti je treba Package Security Identifier – SID – in Client Secret. Po uspešni registraciji se je treba avtentificirati pri WNS-storitvi tako, da pošljemo zahtevo HTTPS na naslednji način:

```
POST /accesstoken.srf HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: https://login.live.com
Content-Length: 211

grant_type=client_credentials&client_id=ms-app%3a%2f%2fS-1-15-2-2972962901-
2322836549-3722629029-1345238579-3987825745-2155616079-
650196962&client_secret=Vex8L9WOFZuj95euaLrvSH7XyoDhLJc7&scope=notify.window
s.com
```

Ko smo vse zgoraj naštete korake uspešno zaključili, lahko nemoteno pošiljamo potisna obvestila mobilni aplikaciji prek WNS-storitev.

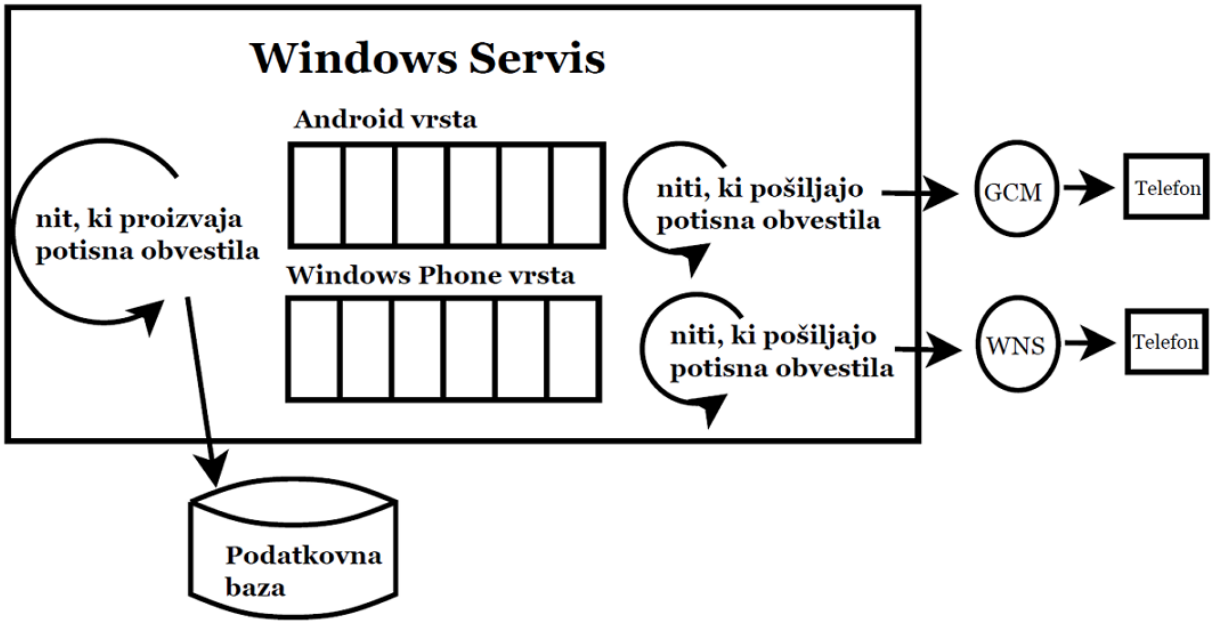
WNS deluje na podoben način kot GCM. Mobilna aplikacija zahteva kanal za potisna obvestila od operacijskega sistema Windows Phone, ki pridobi enoličen identifikator od WNS-storitve, ki ga posreduje mobilni aplikaciji. Mobilna aplikacija mora sama poskrbeti za varen prenos identifikatorja do aplikacijskega strežnika. Ko želi aplikacijski strežnik poslati potisno obvestilo do odjemalske aplikacije, to stori tako, da pošlje zahtevo do WNS-storitev z vključenim identifikatorjem. WNS dobi zahtevo, ki jo preusmeri do mobilne aplikacije.

2.3. Strežnik

Aplikacijski strežnik je namenjen upravljanju, generiranju in pošiljanju potisnih obvestil. Razdeljen je na dve aplikaciji, in sicer na Windows servis in spletno aplikacijo. Windows servis skrbi za branje in pošiljanje potisnih obvestil, spletna aplikacija pa je namenjena administraciji odjemalcev in generiranju potisnih obvestil.

2.3.1 Servis

Arhitektura Windows servisa je prikazana na spodnji sliki:



Slika 2.14: Arhitektura servisa za pošiljanje potisnih obvestil

Pri vsakem zagonu aplikacijskega strežnika se Windows servis avtomatsko zažene. Pri samem zagonu Windows servis inicializira niti, ki so namenjene samemu proizvajanju potisnih obvestil, in niti, ki so namenjene pošiljanju oziroma potrošnji potisnih obvestil ter skupno vrsto za vsak operacijski sistem, v katero se hkrati dodaja in jemlje potisna obvestila. Servisu

je možno poljubno nastavljati število niti tistih, ki proizvajajo, in tistih, ki porabljajo potisna obvestila. Privzeto servis nastavi eno nit za proizvajanje in pet niti za potrošnjo za vsako posamično vrsto. Servis na določen časovni interval preverja nova sporočila. Časovni interval se lahko prek nastavitev spremeni, privzeta vrednost pa je 60 sekund. Vsa neposlana potisna obvestila premakne v vrsto, kjer čakajo na nadaljnje procesiranje. Preden se potisna obvestila postavi v vrsto, se preveri vse potrebne podatke za generiranje potisnega obvestila. Potrebni podatki so:

- eden ali več naslovnikov,
- naslov,
- besedilo in
- slika.

V primeru manjkajočih podatkov se potisno obvestilo ne pošlje. Na podlagi naslovnika se kreira potisno obvestilo določenega tipa (Android ali Windows Phone), ki pripada točno določeni vrsti za pošiljanje. Potisnih obvestil za operacijski sistem Android ne boste našli v vrsti za operacijski sistem Windows Phone, saj niti, namenjene pošiljanju potisnih obvestil Windows Phone, takega potisnega obvestila ne bi znale poslati. To omogočimo na naslednji način:

```
public void QueueNotification<TNotification>(TNotification notification) where TNotification : Notification
{
    var services = GetServices<TNotification>().ToList();

    if (services == null || !services.Any())
        throw new IndexOutOfRangeException(string.Format("no registered services. Type: {0}", typeof(Notification)));

    foreach (var s in services)
        s.QueueNotification(notification);
}

1 reference
public IEnumerable<IService> GetServices<TNotification>()
{
    var type = typeof(TNotification);

    return from sr in _serviceRegistrations
        where sr.NotificationType == type
        select sr.Service;
}
```

Slika 2.15: Implementacija vrste za pošiljanje potisnih obvestil

Ko je potisno obvestilo postavljeno v vrsto, čaka na pošiljanje. Niti, namenjene potrošnji oziroma pošiljanju potisnih obvestil, jemljejo eno po eno potisno obvestilo iz vrste in ga poskušajo poslati.

```
private void DoWork(string workerId, IChannel channel, CancellationTokenSource cancelTokenSource)
{
    Console.WriteLine("{0} DoWork entering..", workerId);

    while (!cancelTokenSource.IsCancellationRequested)
    {

        INotification notification;

        if (_notifications.TryDequeue(out notification))
        {
            Console.WriteLine("{0} DoWork sending..", workerId);
            channel.SendNotification(notification, _sendNotificationCallbackDelegate);
        }

        if (notification == null)
        {
            _manualResetEventSlim.WaitOne();
        }

        Console.WriteLine("{0} DoWork finish..", workerId);
    }
}
```

Slika 2.16: Implementacija niti za pošiljanja potisnega obvestila

```
public void SendNotification(INotification notification, SendNotificationCallbackDelegate callback)
{
    var msg = notification as AndroidNotification;

    if (msg != null)
    {
        Console.WriteLine("Sending notification id: {0}", msg.Tag);

        var webReq = (HttpWebRequest)WebRequest.Create(_settings.GcmUrl);
        webReq.Method = "POST";
        webReq.ContentType = "application/json";
        webReq.UserAgent = "(version: " + "assemblyVerison.ToString()" + ")";
        webReq.Headers.Add("Authorization: key=" + _settings.SenderAuthToken);

        webReq.BeginGetRequestStream(RequestStreamCallback, new AndroidAsyncParameters
        {
            Callback = callback,
            WebRequest = webReq,
            WebResponse = null,
            Message = msg,
            SenderAuthToken = _settings.SenderAuthToken,
            SenderId = _settings.SenderId,
            ApplicationId = _settings.ApplicationIdPackageName
        });
    }
}
```

Slika 2.17: Implementacija pošiljanja potisnega obvestila

Največja razlika pri pošiljanju potisnih obvestil Android ali Windows Phone je pri obliki zahtevka. GCM zahteva obliko zahtevka JSON, WNS pa zahteva obliko zahtevka XML.

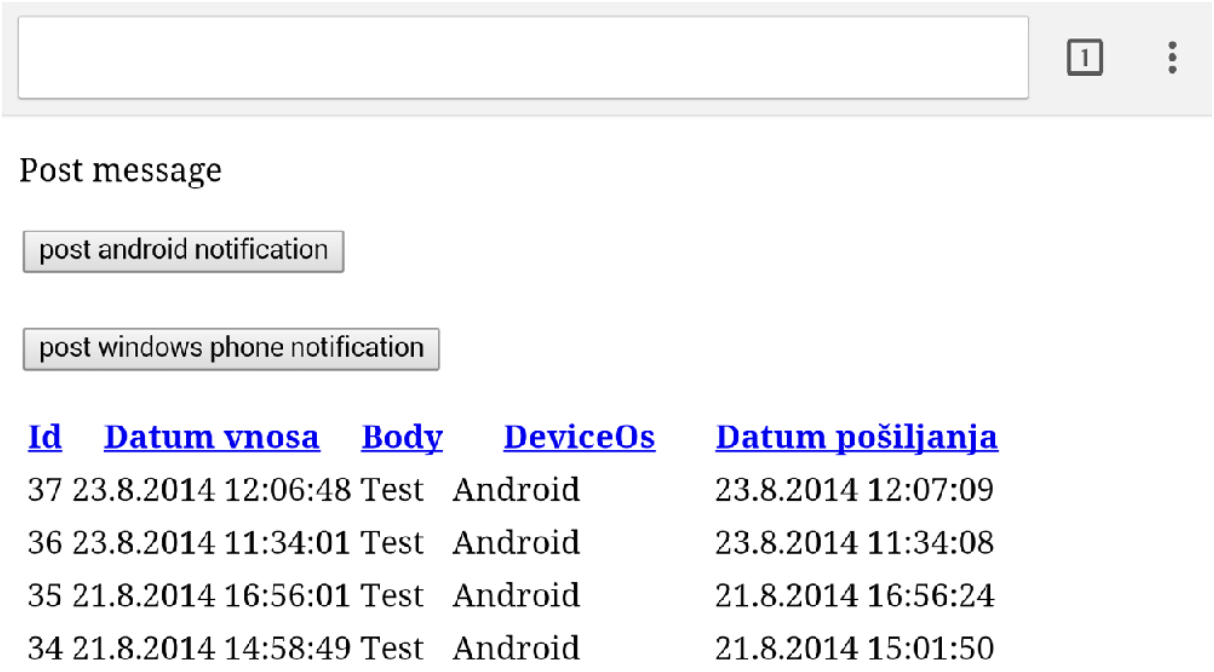
JSON notacija	XML
{	<?xml version=\"1.0\" encoding=\"utf-8\"?>
"collapse_key":5,	<wp:Notification xmlns:wp=\"WPNotification\">
"registration_id":"521771255",	<wp:Toast>
"data":{	<wp:Text1> naslov </wp:Text1>
"title":"naslov",	<wp:Text2> besedilo </wp:Text2>
"besedilo":"besedilo"	<wp:Param> UI razred</wp:Param>
}	</wp:Toast>
}	</wp:Notification>

Tabela 2.1: Razlika med sporočilom GCM in WNS

Vsako sporočilo je v asinhronem načinu poslano do ponudnika potisnih obvestil. Tako ni treba servisu čakati odziva na našo zahtevo, in lahko nadaljujejo s preostalim pošiljanjem potisnih obvestil. Ko je odziv na našo zahtevo pripravljen oz. možen za prevzem, ga servis prevzame in obdela. Vsak odgovor ponudnika sporočila je zapisan v podatkovno bazo za vsako potisno obvestilo posebej. Na tak način lahko zgradimo zgodovino za vsako potisno obvestilo.

2.3.2 Spletna aplikacija

Spletna aplikacija je zgrajen iz dveh modulov. Prvi modul je namenjen administraciji potisnih obvestil, drugi modul pa administraciji pametnih telefonov oziroma mobilnih aplikacij. Modul za administracijo potisnih obvestil omogoča generiranje potisnega obvestila za operacijska sistema Android ali Windows Phone, pregled preteklih obvestil in njihov status.



Slika 2.18: Videz spletnega vmesnika

Modul, namenjen administraciji mobilnih aplikacij, je implementiran v obliki spletnega servisa. Ta omogoča telefonu komunikacijo prek protokola http v obliki notacije JSON. Modul ima implementirano metodo za registracijo telefona na potisna obvestila in odjavo od prejemanja teh obvestil. Modul je možno razširiti tako, da bi hranil podatke o geolokaciji telefona oziroma uporabnika. Na podlagi geolokacije bi lahko obveščali uporabnike o prireditvah, zanimivih lokacijah glede na njihovo trenutno lokacijo – segmentacija uporabnikov po lokaciji.

```
[HttpPost]
0 references
public HttpResponseMessage RegisterDeviceForNotifications(string regId, string os = "unknown", string osVersion = "unknown")
{
    try
    {
        if (CheckRegId(regId))
        {
            return Request.CreateResponse<string>(System.Net.HttpStatusCode.Created, "Device already registered.");
        }

        SqlConnection connection = new SqlConnection();
        connection.ConnectionString = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;

        connection.Open();

        SqlCommand command = connection.CreateCommand();
        StringBuilder sb = new StringBuilder();

        sb.Append(string.Format("INSERT INTO MD_DEVICES (DeviceRegistrationId, DeviceOs, DeviceOsVersion) VALUES ('{0}', '{1}'

        command.CommandText = sb.ToString();
        command.ExecuteNonQuery();

        return Request.CreateResponse<string>(System.Net.HttpStatusCode.Created, "Device successfully registered.");
    }
    catch (Exception e)
    {
        var response = Request.CreateResponse<string>(System.Net.HttpStatusCode.Created, e.ToString());
        return response;
    }
}
```

Slika 2.19: Implementacija registracije mobilne aplikacije

Metodo se pokliče na naslednji način:

http:// ip strežnika /api/device?regId=««&os=««&osVersion=««

2.3.3 Podatkovna zbirka

Ko je potisno obvestilo kreirano in postavljeno v čakalno vrsto za pošiljanje, je zelo pomembno, da so vsi podatki shranjeni v podatkovni bazi. Vrste, ki so kreirane za vsak servis posebej (GCM, WNS), niso obstojne, zato se podatki, shranjeni v vrsti, pri vsakem izpadu električne energije izgubijo. Zelo nehvaležno bi bilo, da bi znova in znova pošiljali enaka potisna obvestila enakim uporabnikom, zato je treba imeti sistem, ki omogoča obstojno hrambo podatkov, kot so podatkovne baze. Ta baza, ki je bila izbrana za hrambo potisnih obvestil, je Microsoft SQL Server, saj je v zadnjih letih pokazala, da se lahko kosa z ostalimi konkurenti na trgu, kot sta Oracle in IBM, in zaradi zelo dobre integracije z obstoječimi razvojnimi Microsoftovimi orodji.

Podatkovni model zajema tri tabele, in sicer tabelo devices, messages in device_message. Tabela devices hrani vse podatke o napravah oziroma telefonih.

```
CREATE TABLE [dbo].[devices] (
   [Id]                BIGINT        IDENTITY (1, 1) NOT NULL,
   [DeviceRegistrationId] NVARCHAR (500) NOT NULL,
   [DeviceOs]          NVARCHAR (500) NOT NULL,
   [DeviceOsVersion]   NVARCHAR (500) NOT NULL,
   [MobilePhone]       NVARCHAR (50)  NULL,
   [InsertTs]          DATETIME      DEFAULT (getdate()) NOT NULL,
   PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

Primer podatkov v tabeli devices:

Id	DeviceRegistrationId	DeviceOs	DeviceOsVersi...	MobilePhone	InsertTs
1	http://s.notify.live.net/u/1/db3/HmQAAACVmTn4YyaZcLAZsT9BE7JxPNkjj0da_c9hqjiRLRisr6K49Vy4xi3HeXK1QY7ma3v7jpJ3mP62VKyKrbxGWbLY/d2IuZG93c3Bob25l...	Windows Phone	8		3.7.2014 19:38:06
2	APA91bHU-6MN--Sevss05RApRezcQDn9l4dX4WGNzRmwwTPmMyw9ASI8_2waGSO2uoXz3uE1c_k3orj5_ec6WMPGaTj3Pissx8O0bn6agmkrR8T-dhAbAp3K7tSfvx_7osU...	Android	4.2		20.7.2014 11:53:00

Slika 2.20: Primer podatkov v tabeli devices

Tabela messages hrani vsa potisna obvestila.

```
CREATE TABLE [dbo].[messages] (
   [Id]                BIGINT        IDENTITY (1, 1) NOT NULL,
   [MsgHeader]         NVARCHAR (500) NOT NULL,
   [MsgBody]           NVARCHAR (500) NOT NULL,
   [ForDeviceOs]       NVARCHAR (500) NOT NULL,
   [ForDeviceOsVersion] NVARCHAR (500) NOT NULL,
   [InsertTs]          DATETIME      DEFAULT (getdate()) NOT NULL,
   PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

Primer podatkov v tabeli messages:

Id	MsgHeader	MsgBody	ForDeviceOs	ForDeviceOsVersion	InsertTs
1	Lorem ipsum	Lorem ipsum dolor sit amet.	Windows Phone	8	3.7.2014 19:38:06
9	Lorem ipsum	Lorem ipsum dolor sit amet.	Android	4.2	3.7.2014 20:38:06

Slika 2.21: Primer podatkov v tabeli messages

Tabela device_message hrani zgodovino vseh poslanih potisnih obvestil. V njej so shranjeni podatki o uspešnem posredovanju potisnega obvestila do uporabnika s strani ponudnika potisnih obvestil (GCM, WNS). Podatek je shranjen v polju ReadTs. Polje SendTs hrani podatek o času, kdaj je potisno obvestilo zapustilo aplikacijski strežnik in bilo prejeto na strani ponudnika. V primeru izpada električne energije se vse čakalne vrste pri ponovnem zagonu servisa napolnijo z vsemi potisnimi obvestili, ki nimajo nastavljenega polja SendTs.

```
CREATE TABLE [dbo].[device_message] (
   [Id]       BIGINT   IDENTITY (1, 1) NOT NULL,
   [MsgId]    BIGINT   NOT NULL,
   [DeviceId] BIGINT   NOT NULL,
   [InsertTs] DATETIME DEFAULT (getdate()) NOT NULL,
   [ReadTs]   DATETIME NULL,
   [SendTs]   DATETIME NULL,
   PRIMARY KEY CLUSTERED ([Id] ASC),
   FOREIGN KEY ([DeviceId]) REFERENCES [dbo].[MD_DEVICES] ([Id]),
   FOREIGN KEY ([MsgId]) REFERENCES [dbo].[MD_MESSAGES] ([Id])
);
```

Primer podatkov iz tabele device_message:

Id	MsgId	DeviceId	InsertTs	ReadTs	SendTs
1	1	1	3.7.2014 19:38:06	20.7.2014 22:05:32	20.7.2014 21:24:30

Slika 2.22: Primer podatkov v tabeli device_message

Poglavje 3 Testiranje programske rešitve

Testiranje programske opreme je postopek pridobivanja podatkov o kakovosti izdelka ali storitve, ki se preizkuša. Na tak način lahko ugotovimo tveganja, ki jih prinaša izdelek ali storitev. Testiranje programske opreme zajema zagon izdelka ali storitve pod različnimi pogoji. V primeru testiranja strežnika za učinkovito obveščanje mobilnih aplikacij se bomo usmerili na porabljen čas pošiljanja potisnih obvestil. Izvedli bomo tri cikle testiranja. Pri prvem ciklu bomo skušali poslati 10.000 potisnih obvestil, pri drugem bomo skušali poslati 100.000 potisnih obvestil, v zadnjem, tretjem ciklu pa bomo poslali 1.000.000 potisnih obvestil.

3.1 Priprava okolja

Za uspešno testiranje strežnika za učinkovito obveščanje mobilnih aplikacij je bil ustvarjen virtualni strežnik z naslednjimi lastnostmi:

- Windows Server 2008 R2 (x64),
- RAM 4GB,
- CPU 2X3,2Ghz (AMD Phenom X4 840).

Na strežnik sta bila nameščena MS SQL Server 2008 in Windows service za pošiljanje potisnih obvestil.

Na strežniku SQL je bilo treba kreirati podatkovno bazo, ki je bila izdelana tako, da smo zagnali skripto iz prejšnjega poglavja 2.2.3 Podatkovna zbirka. Windows service je bil nameščen prek ukazne vrstice z ukazom:

sc create streznik binPath= "C:\Program Files\StreznikZaObvescanje\streznik.exe" DisplayName= "Streznik za obvescanje" start= auto

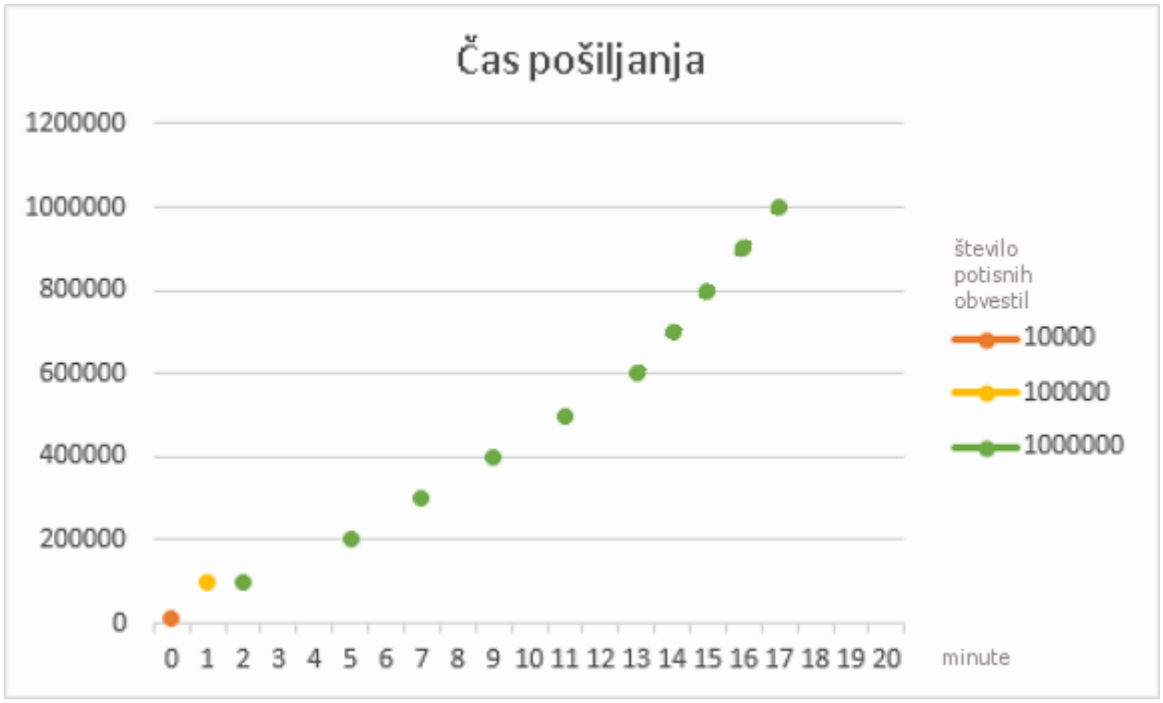
3.2 Meritve

Meritve so potekale v treh ciklih. V prvem ciklu smo spremljali čas pošiljanja 10.000 obvestil, v drugem ciklu čas pošiljanja 100.000 obvestil, v zadnjem, tretjem ciklu pa čas pošiljanja 1.000.000 obvestil.

Ker nismo želeli pošiljati izmišljenih obvestil obstoječim ponudnikom potisnih obvestil (GCM, WNS), so se vsa potisna obvestila pošiljala navideznemu ponudniku. Vsi odgovori na poslana potisna obvestila so bili uspešni.

Čas smo začeli meriti, ko smo začeli potisna obvestila vstavljati v čakalno vrsto. Meritve so se končale, ko se je čakalna vrsta popolnoma izpraznila in v njej ni bilo več nobenih potisnih obvestil za pošiljanje.

Čas, ki ga je strežnik za učinkovito obveščanje mobilnih aplikacij potreboval za pošiljanje 10.000 obvestil, je natanko 8 sekund. V drugem ciklu, kjer je bilo treba poslati 100.000 potisnih obvestil, je potreboval 59 sekund, v zadnjem, tretjem ciklu pri pošiljanju 1.000.000 potisnih obvestil pa je strežnik potreboval 17 minut in 41 sekund.



Slika 0.1: Časi pošiljanja obvestil.

3.3 Ugotovitve

Testiranje strežnika za učinkovito obveščanje mobilnih aplikacij je pokazalo, da strežnik zmore pošiljanje milijon obvestil.

Iz tega lahko sklepamo, da trenutna zasnova strežnika ni dobra za potisna obvestila, ki bi morala biti dostavljena v najkrajšem možnem času do mobilne aplikacije oziroma uporabnika. Da bi se izognili navedenemu problemu, bi bilo treba nadgraditi strežnik za učinkovito obveščanje mobilnih aplikacij tako, da bi implementirali dodatno vrsto, ki bi služila samo za potisna obvestila visoke prioritete. Hkrati bi morali dopolniti obstoječa potisna obvestila z oznako visoka prioriteta, saj trenutna implementacija tega ne omogoča. Zamislite si, da bi morali pošiljati potisna obvestila, ki vsebujejo borzne podatke, kjer igrajo vlogo sekunde. Strežnik za učinkovito pošiljanje bi moral taka potisna obvestila nemudoma poslati.

Poglavje 4 Sklepne ugotovitve

Cilj diplomskega dela sta bili zasnova in izdelava strežnika za učinkovito obveščanje mobilnih aplikacij. V delu je bila predstavljena implementacija strežnika za učinkovito obveščanje mobilnih aplikacij, predstavljeni pa so bili tudi vsi njegovi sestavni deli. Strežnik za učinkovito obveščanje mobilnih aplikacij je sestavljen iz dveh delov; iz dela, ki zajema servis za obveščanje mobilnih aplikacij, in iz spletne aplikacije za administracijo le-teh. Da bi lahko strežnik za učinkovito obveščanje mobilnih aplikacij povezali v celoto, je bilo treba implementirati odjemalsko aplikacijo in se registrirati pri ponudniku potisnih obvestil. Mobilna aplikacija zajema mobilno aplikacijo, ki se registrira pri ponudniku potisnih obvestil, ta pa dodeli mobilni aplikaciji enoličen identifikator, prek katerega prejema potisna obvestila. Trenutna implementacija strežnika za učinkovito obveščanje mobilnih aplikacij komunicira z Googlovim ponudnikom potisnih obvestil – Google Cloud Messaging – in Microsoftovim ponudnikom potisnih obvestil – Windows Notification Services.

Strežniki, namenjeni obveščanju velikih množic, morajo biti zasnovani tako, da jim veliko število potisnih obvestil ne predstavlja težav. Strežnik za učinkovito obveščanje mobilnih aplikacij je bil testiran pri različnih obremenitvah, in sicer je prestal obremenitev enega milijona obvestil. Za pošiljanje tolikšnega števila obvestil je potreboval dobrih sedemnajst minut, iz česar lahko sklepamo, da bi za dva milijona obvestil lahko potreboval slabo uro. Pri obvestilih z visoko prioriteto, ki bi morala biti dostavljena v najkrajšem možnem času, je trenutna implementacija strežnika neprimerna. Za obvestila, ki bi morala biti dostavljena v najkrajšem možnem času, bi lahko strežnik razširili tako, da bi ta najprej pošiljal potisna obvestila z visoko prioriteto, nato pa vsa ostala potisna obvestila. To bi lahko storili tako, da se v vrsto za pošiljanje potisnih obvestil najprej postavi potisna obvestila z visoko prioriteto ali uvede novo vrsto, v kateri so samo potisna obvestila visoke prioritete.

Trenutna implementacija strežnika za učinkovito obveščanje mobilnih aplikacij omogoča učinkovito obveščanje aplikacij Android in Windows Phone oziroma uporabnikov. Strežnik je možno razširiti tako, da bi lahko obveščali mobilne aplikacije ostalih mobilnih operacijskih sistemov, kot sta na primer Ios ali Firefox. Možnosti dodajanja operacijskih sistemov so neomejene. To je ena ključnih prednosti strežnika za učinkovito obveščanje mobilnih aplikacij. Na tak način bi lahko uporabniki obveščali mobilne aplikacije na operacijskih

sistemih, ki jih imajo podprte, poleg trenutno podprtih operacijskih sistemov Android in Windows Phone.

Strežnik bi bilo mogoče razširiti tudi na način, kjer bi si s pomočjo podatkov o geolokaciji obveščal mobilne aplikacije o različnih dogodkih, prireditvah itd. Predstavljajte si, da ste z družino ali prijatelji na dopustu, in iščete restavracijo, v kateri bi radi pojedli nekaj dobrega. Ker pa ste v tujem kraju, restavracij ne poznate. V navedenem primeru bi vas na podlagi vaše geolokacije strežnik za učinkovito obveščanje obvestil o primerni restavraciji.

Literatura

[1] Urban Airship push, Dostopno na:
http://urbanairship.com/products/mobile-app-engagement#push-messages

[2] PushIO, Dostopno na: http://responsys.com/marketing-cloud/products/push-IO

[3] ZeroPush, Dostopno na: https://zeropush.com/

[4] Andrew Troelsen, Pro C# 5.0 and the.NET 4.5 Framework, Apress, 2012

[5] Steven Cheng, Microsoft Windows Communication Foundation 4.0 Cookbook for Developing SOA Applications, Packt publishing, 2010

[6] Stephen Cleary, Concurrency in C# Cookbook, O`Reilly Media Inc, 2014.

[7] Programming ASP.NET 4 MVC, O`Reilly Media Inc, 2012

[8] Andrew Whitechapel, Windows Phone 8 Development Internals, Mirosoft Press, 2012

[9] Alex Davies, Async in C#, O`Reilly Media Inc, 2012.

[10] Reto Meier, Professional Android 4 Application Development, Wiley, 2012

[11] Adam Freeman, Pro .NET 4 Parallel Programming in C#, Apress, 2010

[12] Android SDK, Dostopno na: http://developer.android.com/sdk/index.html

[13] Android Studio, Dostopno na: https://developer.android.com/sdk/installing/studio.html

[14] Android potisna obvestila, Dostopno na:
https://developer.android.com/guide/topics/ui/notifiers/notifications.html

[15] Google Cloud Messaging, Dostopno na:
https://developer.android.com/google/gcm/index.html

[16] Windows Phone potisna obvestila, Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh868259.aspx

[17] WNS, Dostopno na:
http://msdn.microsoft.com/library/windows/apps/hh913756.as